



OpenSPARC™ T1 Processor Design and Verification User's Guide

Sun Microsystems, Inc.
www.sun.com

Part No. 819-5019-10
March 2006, Revision A

Submit comments about this document at: <http://www.sun.com/hwdocs/feedback>

Copyright 2006 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

This document and the product to which it pertains are distributed under licenses restricting their use, copying, distribution, and decompilation. No part of the product or of this document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and in other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, Java, AnswerBook2, docs.sun.com, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and in other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and in other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

U.S. Government Rights—Commercial use. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2006 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, Californie 95054, États-Unis. Tous droits réservés.

Sun Microsystems, Inc. possède les droits de propriété intellectuelle relatifs à la technologie décrite dans ce document. En particulier, et sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plusieurs des brevets américains listés sur le site <http://www.sun.com/patents>, un ou les plusieurs brevets supplémentaires ainsi que les demandes de brevet en attente aux États-Unis et dans d'autres pays.

Ce document et le produit auquel il se rapporte sont protégés par un copyright et distribués sous licences, celles-ci en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a.

Tout logiciel tiers, sa technologie relative aux polices de caractères, comprise, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit peuvent dériver des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux États-Unis et dans d'autres pays, licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, Java, AnswerBook2, docs.sun.com, et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux États-Unis et dans d'autres pays.

Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux États-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface utilisateur graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox dans la recherche et le développement du concept des interfaces utilisateur visuelles ou graphiques pour l'industrie informatique. Sun détient une licence non exclusive de Xerox sur l'interface utilisateur graphique Xerox, cette licence couvrant également les licenciés de Sun implémentant les interfaces utilisateur graphiques OPEN LOOK et se conforment en outre aux licences écrites de Sun.

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DÉCLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES DANS LA LIMITE DE LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE À LA QUALITÉ MARCHANDE, À L'APTITUDE À UNE UTILISATION PARTICULIÈRE OU À L'ABSENCE DE CONTREFAÇON.



Please
Recycle



Adobe PostScript

Contents

- 1. Quick Start 1-1**
 - 1.1 System Requirements 1-1
 - 1.2 EDA Tool Requirements 1-2
 - 1.2.1 EDA Simulation Tools 1-2
 - 1.2.2 EDA Synthesis Tools 1-2
 - 1.3 Running Simulations and Synthesis 1-2
 - 1.3.1 Get the Simulation Files 1-3
 - 1.3.2 Set Up Environment Variables 1-4
 - 1.3.3 Run Your First Regression 1-5
 - 1.3.4 Run Your First Synthesis 1-7
- 2. OpenSPARC T1 Design Implementation 2-1**
 - 2.1 OpenSPARC T1 Design Hierarchy 2-1
 - 2.2 Module Directory Structure 2-3
 - 2.3 Megacells 2-6
 - 2.4 External Interfaces 2-6
- 3. OpenSPARC T1 Verification Environment 3-1**
 - 3.1 OpenSPARC T1 Verification Environment 3-1
 - 3.2 Running a Regression 3-3

3.2.1	To Run a Regression	3-3
3.2.2	What the <code>sims</code> Command Does	3-4
3.2.3	Running Regression With Other Simulators	3-5
3.3	Verification Code	3-5
3.3.1	Verilog Code Used for Verification	3-5
3.3.2	Vera Code Used for Verification	3-6
3.4	PLI Code Used For the Test Bench	3-7
3.4.1	To Compile All PLI Libraries	3-8
3.5	Verification Test File Locations	3-8
3.6	Compiling Source Code for Tools	3-9
4.	OpenSPARC T1 Synthesis	4-1
4.1	Synthesis Flow for the OpenSPARC T1 Processor	4-1
4.2	Synthesis Output	4-3
A.	Design and Verification Manual Pages	A-1
A.1	<code>sims</code>	A-1
A.2	<code>midas</code>	A-15
A.3	<code>goldfinger</code>	A-23
A.4	<code>regreport</code>	A-31
A.5	<code>vlog</code>	A-33

Figures

FIGURE 2-1	OpenSPARC T1 Block Diagram	2-2
FIGURE 2-2	OpenSPARC T1 External Interfaces	2-7
FIGURE 3-1	OpenSPARC T1 Verification Test Bench Overview	3-2

Tables

TABLE 1-1	Disk Space Requirements	1–1
TABLE 1-2	Contents of the <code>OpenSPARCT1</code> Directory	1–3
TABLE 1-3	Environment Variables in <code>.cshrc</code> File	1–4
TABLE 2-1	OpenSPARC T1 Top-Level Modules	2–3
TABLE 3-1	Source Code Types in the Verification Environment	3–2
TABLE 3-2	Details of Regression Groups	3–3
TABLE 3-3	OpenSPARC T1 Verification Test Bench Modules	3–5
TABLE 3-4	PLI Source Code and Object Libraries	3–7
TABLE 3-5	Options for the <code>mkplilib</code> Script	3–8
TABLE 3-6	Verification Test File Directories	3–9
TABLE 4-1	Synthesis Script Details	4–2
TABLE 4-2	Synthesis Output	4–3
TABLE A-1	Environment Variables	A–13

Preface

The *Design and Verification User's Guide* gives an overview of the design hierarchy on the OpenSPARC™ T1 processor. It also describes the files, procedures, and tools needed for running simulations and synthesis on the OpenSPARC T1 processor.

This book covers the following topics:

- Design and Verification implementation overview
- Design and Verification directory and files structure
- System and Electronic Design Automation (EDA) tools required to run simulations and synthesis
- Tools and scripts required to run simulation or complete regressions, including simulation flow
- Synthesis flow and scripts

How This Document Is Organized

[Chapter 1](#) describes quick steps to run simulations after you download the design and verification files from the web site. It also includes system requirements and EDA tools requirements to run simulations and synthesis.

[Chapter 2](#) gives an overview of the OpenSPARC T1 design hierarchy and directory structure.

[Chapter 3](#) gives an overview of the OpenSPARC T1 verification environment implementation and directory structure. The verification environment includes test benches, tests, scripts, and Verilog Programming Language Interface (PLI).

[Chapter 4](#) describes the synthesis flow and synthesis scripts.

[Appendix A](#) has manual pages for regression commands.

Using UNIX Commands

This document might not contain information about basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices. Refer to the following for this information:

- Software documentation that you received with your system
- Solaris™ Operating System documentation, which is at:

<http://docs.sun.com>

Shell Prompts

Shell	Prompt
C shell	<i>machine-name%</i>
C shell superuser	<i>machine-name#</i>
Bourne shell and Korn shell	\$
Bourne shell and Korn shell superuser	#

Typographic Conventions

Typeface*	Meaning	Examples
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. % You have mail.
AaBbCc123	What you type, when contrasted with on-screen computer output	% su password:
<i>AaBbCc123</i>	Book titles, new words or terms, words to be emphasized. Replace command-line variables with real names or values.	Read Chapter 6 in the <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be superuser to do this. To delete a file, type <code>rm filename</code> .

* The settings on your browser might differ from these settings.

Related Documentation

The documents listed as online or download are available at:

<http://www.opensparc.net/>

Application	Title	Part Number	Format	Location
OpenSPARC T1 instruction set	<i>UltraSPARC Architecture 2005 Specification</i>	950-4895-03	PDF	Online
OpenSPARC T1 processor's internal registers	<i>UltraSPARC T1 Supplement to the UltraSPARC Architecture 2005</i>	819-3404-02	PDF	Online
OpenSPARC T1 megacells	<i>OpenSPARC T1 Processor Megacell Specification</i>	819-5016-10	PDF	Download
OpenSPARC T1 signal pin list	<i>OpenSPARC T1 Processor Datasheet</i>	819-5015-10	PDF	Download
OpenSPARC T1 processor J-Bus and SSI interfaces	<i>OpenSPARC T1 Processor External Interface Specification</i>	819-5014-10	PDF	Download

Documentation, Support, and Training

Sun Function	URL
Documentation	http://www.sun.com/documentation/
Support	http://www.sun.com/support/
Training	http://www.sun.com/training/

Third-Party Web Sites

Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can submit your comments by going to:

<http://www.sun.com/hwdocs/feedback>

Please include the title and part number of your document with your feedback:

OpenSPARC T1 Processor Design and Verification User's Guide, part number 819-5019-10

Quick Start

This chapter covers the following topics:

- [System Requirements](#)
- [EDA Tool Requirements](#)
- [Running Simulations and Synthesis](#)

Before you start running simulations or synthesis, make sure you meet system requirements and that you have the required Electronic Design Automation (EDA) tools. Once you download the OpenSPARC T1 tar file from the <http://www.opensparc.com> web site, follow the steps in this chapter to get started and run your first regression on the OpenSPARC T1 design.

1.1 System Requirements

OpenSPARC T1 regressions are currently supported to run only on SPARC® systems running the Solaris 9 or Solaris 10 Operating System.

Disk space requirements are listed in [TABLE 1-1](#).

TABLE 1-1 Disk Space Requirements

Disk Space required	Required for:
1.5 Gbyte	Download, unzip or uncompress, and extract from the tar file
1.6 Gbyte	Run a mini-regression
67 Gbyte	Run a full regression
0.3 Gbyte	Run synthesis
70.4 Gbyte	Total

1.2 EDA Tool Requirements

This section describes the commercial EDA tools required for running simulations for the OpenSPARC T1 processor and synthesizing OpenSPARC T1 Verilog Register Transfer Level (RTL) code.

1.2.1 EDA Simulation Tools

The following EDA tools are required to run Verilog simulations: Verilog Simulator, either VCS or NCVerilog.

- VCS from Synopsys, version 7.1.1R21 or later
- NCVerilog from Cadence, version 5.3.s2 or later

It is permissible to use a Verilog Simulator other than VCS or NCVerilog. See details in [Section 3.2.3, “Running Regression With Other Simulators” on page 3-5](#).

The following EDA tools are optional for running Verilog simulations:

- Vera from Synopsys, version 6.2.8 or later
- Debussy from Novas, version 5.3v19 or later

1.2.2 EDA Synthesis Tools

The following EDA tool is required to perform Verilog RTL synthesis:

- Design Compiler from Synopsys, version X-2005.09 or later

1.3 Running Simulations and Synthesis

This section outlines the steps needed to obtain the simulation tools, set up the simulation environment, run the simulation, and read its log file.

1.3.1 Get the Simulation Files

1. Download the file.

Download the `OpenSPARCT1.tar.bz2` file from the <http://www.opensparc.com> web site. For this procedure's examples, the destination directory is:

```
/home/johndoe/OpenSPARCT1
```

2. Change directories to the directory where you downloaded the file. For example:

```
% cd /home/johndoe/OpenSPARCT1
```

3. Use the `bunzip2` command to unzip the file.

```
% bunzip2 OpenSPARC_1.tar.bz2
```

4. Extract the tar file using the `tar` command.

```
% tar -xvf OpenSPARC_1.tar
```

This step creates the files and subdirectories listed in [TABLE 1-2](#) in your current directory.

TABLE 1-2 Contents of the OpenSPARCT1 Directory

Name	Type	Description
OpenSPARCT1.cshrc	File	File to set up environment variables and paths
README	File	Instructions to set up and run simulations
lib	Directory	Verilog libraries
verif	Directory	Verification directories and files
design	Directory	Verilog RTL for OpenSPARC T1 design
tools	Directory	Tools and scripts needed to run simulations and synthesis
doc	Directory	Documentation in PDF form for the OpenSPARC T1 processor

1.3.2 Set Up Environment Variables

Edit the `OpenSPARCT1.cshrc` file to set the required environment variables as shown in [TABLE 1-3](#):

TABLE 1-3 Environment Variables in `.cshrc` File

Environment Variable	Usage	Example value
DV_ROOT	Running simulations and synthesis	<code>/home/johndoe/OpenSPARCT1</code> (Directory where you ran the tar command above)
MODEL_DIR	Running simulations	<code>/home/johndoe/OpenSPARCT1_model</code> (Directory where you want to run your simulations)
VERA_HOME	Running simulations	<code>/import/EDAtools/vera/vera,v6.2.10/5.x</code> (Directory where Vera is installed)
NOVAS_HOME	Running simulations	<code>/import/EDAtools/debussy/debussy,v5.3v19/5.x</code> (Directory where Debussy is installed)
VCS_HOME	Running VCS simulations	<code>/import/EDAtools/vcs7.1.1R21</code> (Directory where VCS is installed)
NCV_HOME	Running NCVerilog simulations	<code>/import/EDAtools/ncverilog/ncverilog.v5.3.s2/5.x</code> (Directory where NCVerilog is installed)
SYN_HOME	Running synthesis	<code>/import/EDAtools/synopsys/synopsys.vX-2005.09</code> (Directory where Synopsys is installed)
CC_BIN	Compiling PLI code	<code>/usr/dist/pkgs/devpro,v4.2/5.x-sparc/bin</code> (Directory where C++ Compiler binaries are installed)
LM_LICENSE_FILE	Running simulations and synthesis	<code>/import/EDAtools/licenses/synopsys_key:/import/EDAtools/licenses/ncverilog_key</code> (EDA tool license files)

Once you set the environment variables from [TABLE 1-3](#), the `OpenSPARCT1.cshrc` file sets the following environment variables:

- `TRE_ENTRY`
- `TRE_LOG`
- `TRE_SEARCH`
- `ENVDIR`
- `PERL_MODULE_BASE`

The `OpenSPARCT1.cshrc` script also adds the following directories to your `PATH` and path variables:

- `$DV_ROOT/tools/bin`
- `$NCV_HOME/tools/bin`
- `$VCS_HOME/bin`

- \$VERA_HOME/bin
- \$SYN_HOME/sparcOS5/syn/bin
- \$CC_BIN

After completing your `OpenSPARCT1.cshrc` file edits, source it by using the `source` command:

```
% source /home/johndoe/OpenSPARCT1/OpenSPARCT1.cshrc
```

You might want to include the above command in your `~/ .cshrc` file so that the above environment variables are set every time you log in.

1.3.3 Run Your First Regression

The OpenSPARC T1 Design/Verification package comes with two test bench environments: `core1` and `chip8`.

The `core1` environment consists of:

- One SPARC CPU core
- Cache
- Memory
- Crossbar

The `core1` environment does not have an I/O subsystem.

The `chip8` environment consists of:

- A full OpenSPARC T1 chip, including all eight cores
- Cache
- Memory
- Crossbar
- I/O subsystem

Each environment can perform either a mini-regression or a full regression.

To run a regression, use the `sims` command as described in [Section 1.3.3.1, “To Run a Regression” on page 1-6](#). The important parameters for the `sims` command are:

- `-sim_type`: Simulator type

Set this to `vcs` or `ncv`. For example: **`-sim_type=vcs`**

- `-group`: Regression group name

The choices for `-group` are: `core1_mini`, `core1_full`, `chip8_mini`, and `chip8_full`. For example: **`-group=core1_mini`**

1.3.3.1 To Run a Regression

1. Create the `$MODEL_DIR` directory.

```
% mkdir $MODEL_DIR
```

2. Change directory to `$MODEL_DIR`.

```
% cd $MODEL_DIR
```

This is where the simulations are run.

3. Run a mini-regression for the `core1` environment using the VCS simulator.

```
% sims -sim_type=vcs -group=core1_mini
```

This command creates two directories:

- A directory called `core1` under `$MODEL_DIR`. The regression compiles Vera and Verilog code under the `core1` directory. This is the Vera and Verilog “build” directory.
- A directory named with today’s date and a serial number, such as `2006_01_07_0` (the format is `YYYY_MM_DD_ID`) under the current directory where simulations will run. This is the Verilog simulation’s “run” directory. There is one subdirectory under this directory for each diagnostics test.

By default, the simulations are run with Vera. If you do not want to use Vera, add following option to the `sims` command:

```
-novera_build -novera_run
```

4. Once simulations are completed, run the `regreport` command to generate a regression report.

```
% cd run-directory  
% regreport $PWD > report.log
```

Where *run-directory* is the “run” directory created in the above step, such as `2006_01_07_0`.

The `core1_mini` regression has 68 tests. An example of its `report.log` output is shown in [CODE EXAMPLE 1-1](#).

CODE EXAMPLE 1-1 Example report.log Regression Output

=====			
Status:	core1_mini		ALL

PASS:	68		68
FAIL:	0		0
Diag Problem:	0		0
License Problem:	0		0
MaxCycles Hit:	0		0
Socket Problem:	0		0
Timeout:	0		0
LessThreads:	0		0
Simics Problem:	0		0
Performance:	0		0
Killed By Job Q:	0		0
Unknown:	0		0
UnFinished:	0		0
flexlm error:	0		0

Diag Count:	68		68

If your `report.log` file displays a similar status, you have successfully completed running a mini-regression for the OpenSPARC T1 processor.

1.3.4 Run Your First Synthesis

The command to run a synthesis is `rsyn`. For example, to run a synthesis for one of the modules called `efc`, type:

```
% rsyn efc
```

This command runs a synthesis for the `efc` block and creates gate level netlists under the `$DV_ROOT/design/sys/iop/efc/synthesis/gate` directory.

The synthesis flow and scripts are described in more detail in [Chapter 4](#).

OpenSPARC T1 Design Implementation

This chapter gives details on the following topics:

- [OpenSPARC T1 Design Hierarchy](#)
- [Module Directory Structure](#)
- [Megacells](#)
- [External Interfaces](#)

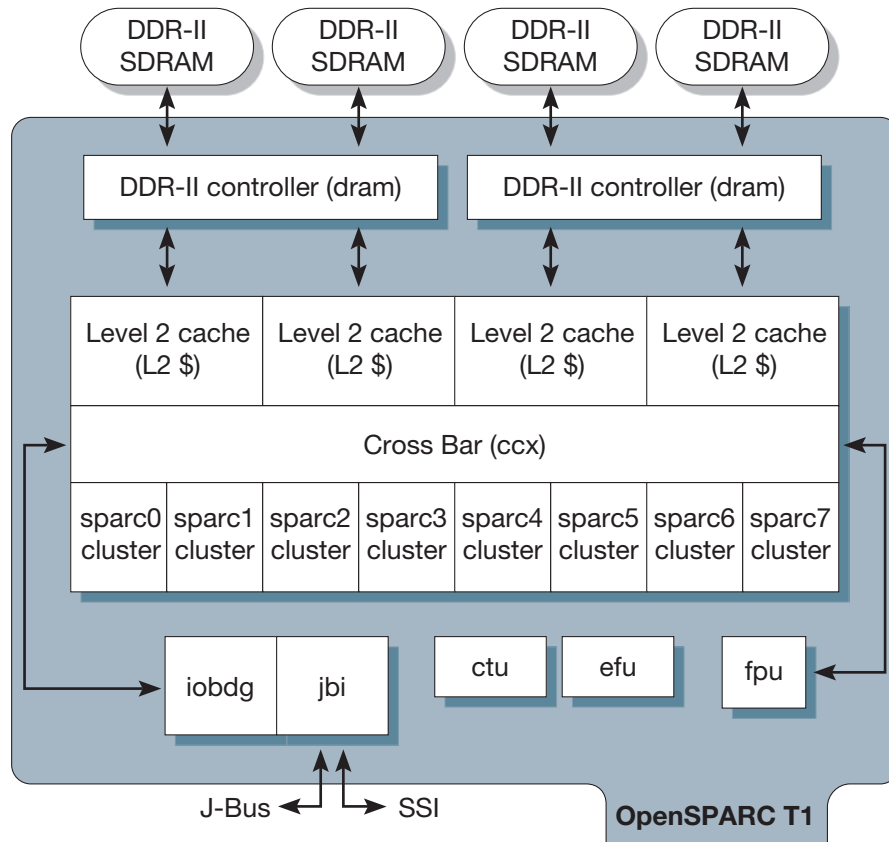
2.1 OpenSPARC T1 Design Hierarchy

The top-level Verilog module for the OpenSPARC T1 processor is called `OpenSPARCT1`. There are various types of design blocks at the top level:

- **Cluster** – A hierarchical block with one or more instances of this block in the design. For example, a SPARC CPU core is called `sparc` and has eight instances at the top level.
- **Pads** – Input, Output and Bi-directional pins on the OpenSPARC T1 processor, including input buffer, output driver, etc. For example, `pad_ddr0` contains pads for DDR bank 0.
- **Repeaters** – Many buffers and repeaters at the top level for signals going to blocks or signals with long traces in the physical implementation, such as `dram0_ddr0_rptr`.
- **Clock** – This includes global clock distribution, including buffers, drivers, repeaters, etc.

A block diagram of the OpenSPARC T1 processor is shown in [FIGURE 2-1](#).

FIGURE 2-1 OpenSPARC T1 Block Diagram



2.2 Module Directory Structure

The Verilog RTL for the OpenSPARC T1 processor is in the `$DV_ROOT/design/sys/iop` directory. All the top-level modules that make up that RTL, and their locations, are listed in TABLE 2-1. You can also browse the Verilog source code on the OpenSPARC web site at <http://www.opensparc.net>.

TABLE 2-1 OpenSPARC T1 Top-Level Modules

Module Name	Type	Number of Instances	Instance Names	Directory Location under \$DV_ROOT/design/sys/iop	Description
ccx	Cluster	1	ccx	ccx	CPU-Cache Cross bar
ctu	Cluster	1	ctu	ctu	Clock and Test Unit
dram	Cluster	2	dram02, dram13	dram	DRAM controller
efc	Cluster	1	efc	efc	e-Fuse Cluster
fpu	Cluster	1	fpu	fpu	Floating Point Unit
iobdg	Cluster	1	iobdg	iobdg	I/O bridge
jbi	Cluster	1	jbi	jbi	J-Bus Interface
scbuf	Cluster	4	scbuf[0-3]	scbuf	L2 \$ buffer
scdata	Cluster	4	scdata[0-3]	scdata	L2 \$ data
sctag	Cluster	4	sctag[0-3]	sctag	L2 \$ tag
sparc	Cluster	8	sparc[0-7]	sparc	SPARC CPU core
pad_ddr0	Pad	1	pad_ddr0	pads	DDR0 pads
pad_ddr1	Pad	1	pad_ddr1	pads	DDR1 pads
pad_ddr2	Pad	1	pad_ddr2	pads	DDR2 pads
pad_ddr3	Pad	1	pad_ddr3	pads	DDR3 pads
pad_efc	Pad	1	pad_efc	pads	efc pads
pad_jbusr	Pad	1	pad_jbusr	pads	J-Bus pads
pad_jbusl	Pad	2	pad_jbusl, pad_dbg	pads	J-Bus pads

TABLE 2-1 OpenSPARC T1 Top-Level Modules (*Continued*)

Module Name	Type	Number of Instances	Instance Names	Directory Location under \$DV_ROOT/design/sy s/iop	Description
pad_misc	Pad	1	pad_misc	pads	Miscellaneous pads
bw_temp_diode	Pad	2	pad_diode0, pad_diode1	analog	Temperature diode pads
bw_ctu_pad_cluster	Pad	1	pad_ctu	analog	CTU pads
ccx_iob_rptr	Repeater	1	ccx_iob_rptr	cmp	ccx repeater
ccx_spc_rpt	Repeater	8	ccx_spc_rpt[0-7]	cmp	ccx repeater
ctu_top_rptr	Repeater	1	ctu_top_rptr	cmp	ctu repeater
ctu_top_rptr2	Repeater	1	ctu_top_rptr2	cmp	ctu repeater
ctu_bottom_rptr	Repeater	1	ctu_bottom_rptr	cmp	ctu repeater
ctu_bottom_rptr2	Repeater	1	ctu_bottom_rptr2	cmp	ctu repeater
dram0_ddr0_rptr	Repeater	1	dram0_ddr0_rptr0	cmp	dram repeater
dram1_ddr1_rptr	Repeater	1	dram1_ddr1_rptr0	cmp	dram repeater
dram2_ddr2_rptr	Repeater	1	dram2_ddr2_rptr0	cmp	dram repeater
dram3_ddr3_rptr	Repeater	1	dram3_ddr3_rptr0	cmp	dram repeater
dram_ddr_pad_rptr	Repeater	2	dram0_ddr0_rptr2, dram2_ddr2_rptr2	cmp	dram repeater
dram_ddr_pad_rptr_south	Repeater	2	dram1_ddr1_rptr2, dram3_ddr3_rptr2	cmp	dram repeater
dram_ddr_rptr	Repeater	2	dram0_ddr0_rptr1, dram2_ddr2_rptr1	cmp	dram repeater
dram_ddr_rptr_south	Repeater	2	dram1_ddr1_rptr1, dram3_ddr3_rptr1	cmp	dram repeater
dram_l2_buf2	Repeater	8	dram_sc_[0-3]_rep1, dram_sc_[0-3]_rep3	cmp	dram repeater
dram_sc_0_rep2	Repeater	1	dram_sc_0_rep2	cmp	dram repeater
dram_sc_1_rep2	Repeater	1	dram_sc_1_rep2	cmp	dram repeater
dram_sc_2_rep2	Repeater	1	dram_sc_2_rep2	cmp	dram repeater
dram_sc_3_rep2	Repeater	1	dram_sc_3_rep2	cmp	dram repeater
ff_dram_sc_bank0	Repeater	1	ff_dram_sc_bank0	cmp	dram repeater

TABLE 2-1 OpenSPARC T1 Top-Level Modules (*Continued*)

Module Name	Type	Number of Instances	Instance Names	Directory Location under \$DV_ROOT/ design/sy s/iop	Description
ff_dram_sc_bank1	Repeater	1	ff_dram_sc_bank1	cmp	dram repeater
ff_dram_sc_bank2	Repeater	1	ff_dram_sc_bank2	cmp	dram repeater
ff_dram_sc_bank3	Repeater	1	ff_dram_sc_bank3	cmp	dram repeater
ff_jbi_sc0_1	Repeater	1	ff_jbi_sc0_1	cmp	jbi repeater
ff_jbi_sc0_2	Repeater	1	ff_jbi_sc0_2	cmp	jbi repeater
ff_jbi_sc1_1	Repeater	1	ff_jbi_sc1_1	cmp	jbi repeater
ff_jbi_sc1_2	Repeater	1	ff_jbi_sc1_2	cmp	jbi repeater
ff_jbi_sc2_1	Repeater	1	ff_jbi_sc2_1	cmp	jbi repeater
ff_jbi_sc2_2	Repeater	1	ff_jbi_sc2_2	cmp	jbi repeater
ff_jbi_sc3_1	Repeater	1	ff_jbi_sc3_1	cmp	jbi repeater
ff_jbi_sc3_2	Repeater	1	ff_jbi_sc3_2	cmp	jbi repeater
iob_ccx_rptr	Repeater	1	iob_ccx_rptr	cmp	iob repeater
iob_jbi_rptr_0	Repeater	1	iob_jbi_rptr_0	cmp	iob repeater
iob_jbi_rptr_1	Repeater	1	iob_jbi_rptr_1	cmp	iob repeater
jbi_l2_buf2	Repeater	4	rep_jbi_sc[0-3]_1	cmp	jbi repeater
rep_jbi_sc0_2	Repeater	1	rep_jbi_sc0_2	cmp	jbi repeater
rep_jbi_sc1_2	Repeater	1	rep_jbi_sc1_2	cmp	jbi repeater
rep_jbi_sc2_2	Repeater	1	rep_jbi_sc2_2	cmp	jbi repeater
rep_jbi_sc3_2	Repeater	1	rep_jbi_sc3_2	cmp	jbi repeater
sc_0_1_dbg_rptr	Repeater	1	sc_0_1_dbg_rptr	cmp	L2 repeater
sc_2_3_dbg_rptr	Repeater	1	sc_2_3_dbg_rptr	cmp	L2 repeater
sctag_cpx_rptr_0	Repeater	1	sctag_cpx_rptr_0	cmp	L2 repeater
sctag_cpx_rptr_1	Repeater	1	sctag_cpx_rptr_1	cmp	L2 repeater
sctag_cpx_rptr_2	Repeater	1	sctag_cpx_rptr_2	cmp	L2 repeater
sctag_cpx_rptr_3	Repeater	1	sctag_cpx_rptr_3	cmp	L2 repeater
sctag_pcx_rptr_0	Repeater	1	sctag_pcx_rptr_0	cmp	L2 repeater
sctag_pcx_rptr_1	Repeater	1	sctag_pcx_rptr_1	cmp	L2 repeater
sctag_pcx_rptr_2	Repeater	1	sctag_pcx_rptr_2	cmp	L2 repeater

TABLE 2-1 OpenSPARC T1 Top-Level Modules (*Continued*)

Module Name	Type	Number of Instances	Instance Names	Directory Location under \$DV_ROOT/design/s/s/iop	Description
sctag_pcx_rpтр_3	Repeater	1	sctag_pcx_rpтр_3	cmp	L2 repeater
sctag_scbuf_rpтр0	Repeater	1	sctag_scbuf_rpтр0	cmp	L2 repeater
sctag_scbuf_rpтр1	Repeater	1	sctag_scbuf_rpтр1	cmp	L2 repeater
sctag_scbuf_rpтр2	Repeater	1	sctag_scbuf_rpтр2	cmp	L2 repeater
sctag_scbuf_rpтр3	Repeater	1	sctag_scbuf_rpтр3	cmp	L2 repeater
spc_pcx_buf	Repeater	8	buf_pcx_[0-7]	sparc	Buffer
bw_clk_gl	Clock	1	bw_clk_gl	analog	Global clock distribution and buffers
bw_clk_gl_rstce_rtl	Clock	1	flop_rpтрs	analog	Global clock buffers and repeaters

2.3 Megacells

The OpenSPARC T1 design contains many megacells, which are custom blocks for static random access memory (SRAMs), translation lookaside buffer (TLB), TAGs, Level 2 Cache, and so on. These megacells are instantiated in the top-level clusters. The detailed descriptions of all megacells, including their function descriptions, I/O lists, block diagrams, and timing diagrams, are in the *OpenSPARC T1 Megacell Specification*.

2.4 External Interfaces

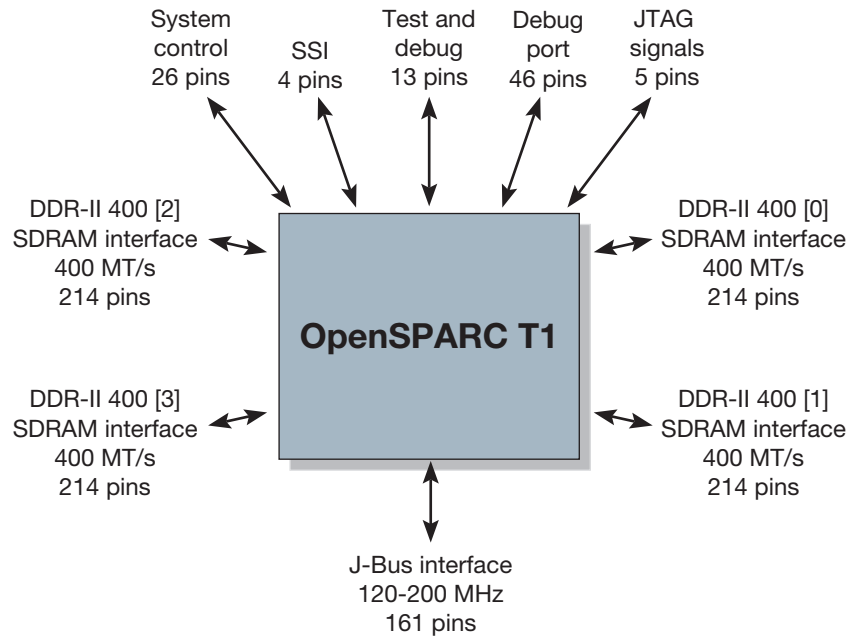
The OpenSPARC T1 processor has the following external interfaces:

- Four DDR-II interfaces
- J-Bus
- SSI - Serial System Interface
- JTAG - IEEE 1149.1 interface

- System control
- Test and debug
- Debug port

The block diagram of external interfaces is shown in [FIGURE 2-2](#).

FIGURE 2-2 OpenSPARC T1 External Interfaces



Total: 1111 Signal Pins

OpenSPARC T1 Verification Environment

This chapter describes the following topics:

- [OpenSPARC T1 Verification Environment](#)
- [Running a Regression](#)
- [Verification Code](#)
- [PLI Code Used For the Test Bench](#)
- [Verification Test File Locations](#)

3.1 OpenSPARC T1 Verification Environment

The OpenSPARC T1 verification environment is a highly automated environment. With a simple command, you can run the entire regression suite for the OpenSPARC T1 processor, containing thousands of tests. With a second command, you can check the results of the regression.

The OpenSPARC T1 Design and Verification package comes with two test bench environments: `core1` and `chip8`.

The `core1` environment consists of:

- One SPARC CPU core
- Cache
- Memory
- Crossbar

The `core1` environment does not have an I/O subsystem.

The chip8 environment consists of:

- A full OpenSPARC T1 chip, including all eight cores
- Cache
- Memory
- Crossbar
- I/O subsystem

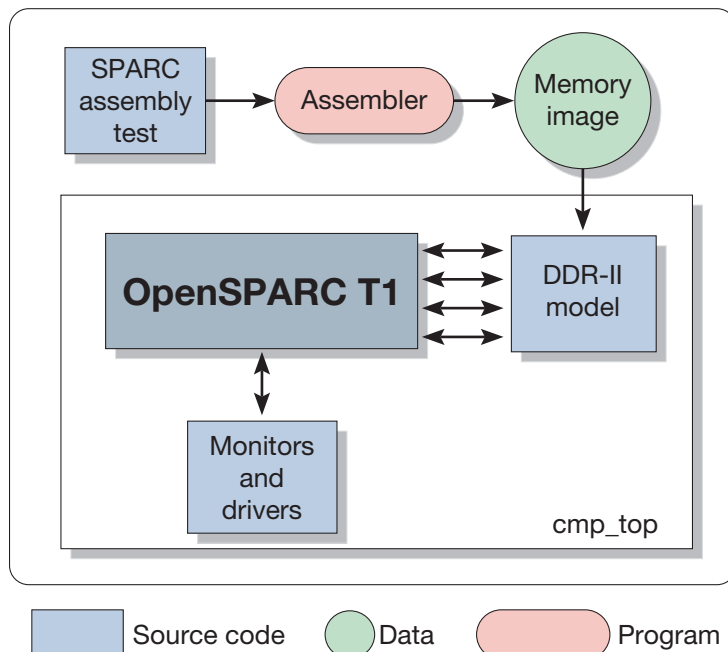
The verification environment uses source code in various languages. [TABLE 3-1](#) shows a summary of the types of source code and their uses.

TABLE 3-1 Source Code Types in the Verification Environment

Source Code Language	Used for:
Verilog	Chip design, test bench drivers, and monitors.
Vera	Test bench drivers, monitors, and coverage objects. Use of Vera is optional.
PERL	Scripts for running simulations and regressions.
C and C++	PLI (Programming Language Interface) for Verilog.
SPARC Assembly	Verification tests.

The block diagram for the verification test bench is in [FIGURE 3-1](#).

FIGURE 3-1 OpenSPARC T1 Verification Test Bench Overview



The top-level module for the test bench is called `cmp_top`. The same test bench is used for both the `core1` and `chip8` environments with compile-time options.

3.2 Running a Regression

For each environment, there is a mini-regression and a full regression. [TABLE 3-2](#) describes the regression groups.

TABLE 3-2 Details of Regression Groups

Regression Group name	Environment	No. of Tests	Disk space needed to run (Mbyte)
core1_mini	core1	68	60
core1_full	core1	900	6,582
chip8_mini	chip8	492	1,517
chip8_full	chip8	3789	60,458

3.2.1 To Run a Regression

1. Run the `sims` command with your chosen parameters.

For instance, to run a mini-regression for the `core1` environment using the VCS simulator, set up the `sims` command as follows:

```
% sims -sim_type=vcs -group=core1_mini
```

To run regressions on multiple groups at the same time, specify multiple `-group=` parameters at the same time. For a complete list of command-line options for the `sims` command, see [Appendix A](#).

2. Run the `regreport` command to get a summary of the regression.

```
% regreport $PWD/2006_01_25_0 > report.log
```

3.2.2 What the `sims` Command Does

When running a simulation, the `sims` command performs the following steps:

1. Compiles the design into the `$MODEL_DIR/core1` or `$MODEL_DIR/chip8` directory, depending on which environment is being used.
2. Creates a directory for regression called `$PWD/DATE_ID`, where `$PWD` is your current directory, `DATE` is in `YYYY_MM_DD` format, and `ID` is a serial number starting with 0. For example, for the first regression on Jan 25, 2006, a directory called `$PWD/2006_01_26_0` is created. For the second regression run on the same day, the last ID is incremented to become `$PWD/2006_01_26_1`.
3. Creates a `master_diaglist.regression_group` file under the above directory. such as `master_diaglist.core1_mini` for the `core1_mini` regression group. This file is created based on diaglists under the `$DV_ROOT/verif/diag` directory.
4. Creates a subdirectory with the test name under the regression directory created in step 2 above.
5. Creates a `sim_command` file for the test based on the parameters in the diaglist file for the group.
6. Executes `sim_command` to run a Verilog simulation for the test. If the `-sas` option is specified for the test, it also runs the SPARC Architecture Simulator (SAS) in parallel with the Verilog simulator. The results of the Verilog simulation are compared with the SAS results after each instruction.

The `sim_command` command creates many files in the test directory. Following are the sample files in the test directory:

<code>diag.ev</code>	<code>diag.s</code>	<code>l2way.log</code>	<code>perf.log</code>
<code>sas.log.gz</code>	<code>sims.log</code>	<code>symbol.tbl</code>	<code>sim.perf.log</code>
<code>diag.exe.gz</code>	<code>efuse.img</code>	<code>midas.log</code>	<code>sim_command</code>
<code>status.log</code>	<code>sim.log.gz</code>		

The `status.log` file has a summary of the status, where the first line contains the name of the test and its status (PASS/FAIL).

```
Diag: xor_imm_corner:model_core1:core1_full:0    PASS
```

7. Repeats steps 4 to 6 for each test in the regression group.

3.2.3 Running Regression With Other Simulators

To use a Verilog simulator other than VCS or NCVerilog, use following options for the sims command:

```
-sim_type="Your simulator name"
-sim_build_cmd="Your simulator command to build/compile RTL"
-sim_run_cmd="Your simulator command to run simulations"
-sim_build_args="Arguments to build/compile"
-sim_run_args="Arguments to run simulations"
```

You only need to specify the `sim_type`, `sim_build_cmd`, and `sim_run_cmd` options once. You can specify `sim_build_args` and `sim_run_args` multiple times to specify multiple argument options.

3.3 Verification Code

This section outlines Verilog and Vera code structures and locations.

3.3.1 Verilog Code Used for Verification

There are various test bench drivers and monitors written in Verilog. A list of all Verilog modules, the location of the source code, and descriptions is in [TABLE 3-3](#). All verification Verilog files are in the `$DV_ROOT/verif/env` directory.

TABLE 3-3 OpenSPARC T1 Verification Test Bench Modules

Module Name	Type	Number of instances	Instance Names	Directory Location under \$DV_ROOT/verif/env	Description
OpenSPARCT1	Chip	1	iop	\$DV_ROOT/design/sys/iop/rtl	OpenSPARC T1 top level
bw_sys	Driver	1	bw_sys	cmp	SSI bus driver
cmp_clk	Driver	1	cmp_clk	cmp	Clock driver
cmp_dram	Model	1	cmp_dram	cmp	DRAM modules
cmp_mem	Driver	1	cmp_mem	cmp	Memory tasks
cpx_stall	Driver	1	cpx_stall	cmp	CPX stall

TABLE 3-3 OpenSPARC T1 Verification Test Bench Modules (*Continued*)

Module Name	Type	Number of instances	Instance Names	Directory Location under \$DV_ROOT/verif/env	Description
dbg_port_chk	Monitor	1	dbg_port_chk	cmp	Debug port checker
dffrl_async	Driver	4	flop_dds[0-3]_oe	\$DV_ROOT/design/sys/iop/common/rtl	Flip-flop
err_inject	Driver	1	err_inject	cmp	Error Injector
jbus_monitor	Monitor	1	jbus_monitor	iss/pli/jbus_mon/rtl	J-Bus Monitor
jp_sjm	Driver	2	j_sjm_4, j_sjm_5	iss/pli/sjm/rtl	J-Bus Driver
monitor	Monitor	1	monitor	cmp	Various monitors
one_hot_mux_mon	Monitor	1	one_hot_mux_mon	cmp	Hot mux monitor
pcx_stall	Driver	1	pcx_stall	cmp	PCX stall
sas_intf	SAS	1	sas_intf	cmp	SAS interface
sas_tasks	SAS	1	sas_tasks	cmp	SAS tasks
slam_init	Driver	1	slam_init	cmp	Initialization tasks
sparc_pipe_flow	Monitor	1	sparc_pipe_flow	cmp	SPARC pipe flow monitor
tap_stub	Driver	1	tap_stub	cmp	JTAG driver

3.3.2 Vera Code Used for Verification

Two types of Vera code are included in the OpenSPARC T1 verification environment:

- Test bench driver and Monitor Vera code
- Vera Object coverage Vera code

Vera code is in the \$DV_ROOT/verif/env/cmp/vera directory. Each object coverage module has a corresponding subdirectory. Following is a list of Vera object coverage modules:

cmpmss_coverage	dram_coverage	exu_coverage	fpu_coverage
lsu_coverage	mt_coverage	tlu_coverage	coreccx_coverage
err_coverage	ffu_coverage	ifu_coverage	mmu_coverage
spu_coverage	tso_coverage		

Object coverage Vera code for jbi is in the `$DV_ROOT/verif/env/iss/vera/jbi_coverage` directory. Object coverage Vera code is only used for the `chip8_cov` regression groups.

3.4 PLI Code Used For the Test Bench

Verilog's PLI (Programming Language Interface) is used to drive and monitor the simulations of the OpenSPARC T1 design. There are eight different directories for PLI source code. Some PLI code is in C language, and some is in C++ language. The object libraries for the VCS simulator and NC-Verilog simulator are included for the PLI code in the `$DV_ROOT/tools/SunOS/sparc/lib` directory. [TABLE 3-4](#) gives the details of PLI code directories, VCS libraries, and NC-Verilog libraries.

TABLE 3-4 PLI Source Code and Object Libraries

PLI name	Source code location under \$DV_ROOT	VCS object library name	NC-Verilog object library name	Description
iop	tools/pli/iop	libiob.a	libiob_ncv.so	Monitors and drivers
mem	tools/pli/mem	libmem_pli.a	libmem_pli_ncv.so	Memory read/write
socket	tools/pli/socket	libsocket_pli.a	libsocket_pli_ncv.so	Sockets to SAS
utility	tools/pli/utility	libutility_pli.a	libutility_ncv.so	Utility functions
common	verif/env/iss/pli/common/c	libjpccommon.a	libjpccommon_ncv.so	Common PLI functions
jbus_mon	verif/env/iss/pli/jbus_mon/c	libjbus_mon.a	libjbus_mon_ncv.so	J-Bus Monitor
monitor	verif/env/iss/pli/monitor/c	libmonitor.a	libmonitor_ncv.so	Various
sjm	verif/env/iss/pli/sjm/c	libsjm.a	libsjm_ncv.so	J-Bus Driver

VCS object libraries are statically linked libraries (.a files) which are linked when VCS compiles the Verilog code to generate a `simv` executable. NC-Verilog object libraries are dynamically loadable libraries (.so files) which are linked dynamically while running the simulations.

Makefiles are provided to compile PLI code. There is a `makefile` file under each PLI directory to create a static object library (`.a` file). There is a `makefile.ncv` file under each PLI directory to create a dynamic object library.

3.4.1 To Compile All PLI Libraries

To compile all PLI libraries, run the `mkplilib` script. This script has three options as listed in [TABLE 3-5](#).

TABLE 3-5 Options for the `mkplilib` Script

Option	Used for
<code>vcs</code>	Compiling PLI libraries for VCS
<code>ncverilog</code>	Compiling PLI libraries for NC-Verilog
<code>clean</code>	Deleting all PLI libraries

- **Compile PLI libraries with your chosen option.**

For example, to compile PLI libraries in VCS, type the following:

```
% mkplilib vcs
```

Either version of this procedure, VCS or NC_Verilog, compiles C/C++ code, creates static or dynamic libraries, and copies them to the `$DV_ROOT/tools/SunOS/sparc/lib` directory.

3.5 Verification Test File Locations

The verification or diagnostics tests (diags) for the OpenSPARC T1 processor are written in SPARC assembly language (the file names have a `.s` extension). Some diags require command files for a J-Bus Driver. Those command files are named `sjm_4.cmd` and `sjm_5.cmd`. Some diagnostics test cases in SPARC assembly are automatically generated by Perl scripts.

The main diaglist for core1 is core1.diaglist. The main diaglist for chip8 is chip8.diaglist. These main diaglists for each environment also include many other diaglists. The locations of various verification test files are listed in [TABLE 3-6](#).

TABLE 3-6 Verification Test File Directories

Directory	Contents
\$DV_ROOT/verif/diag	All diagnostics, various diagnostic list files with the extension .diaglist.
\$DV_ROOT/verif/diag/assembly	Source code for SPARC assembly diagnostics. More than 2000 assembly test files.
\$DV_ROOT/verif/diag/efuse	EFuse cluster default memory load files.

3.6 Compiling Source Code for Tools

To compile source code for some Sun tools used for the OpenSPARC T1 processor, use the mktools script. The tools source code is located in the \$DV_ROOT/tools/src directory.

The mktools script compiles the source code and copies the binaries to \$DV_ROOT/tools/<Operating System>/<Processor Type> directory, where:

- <Operating System> is defined by the uname -s command
- <Processor Type> is defined by the uname -p command

OpenSPARC T1 Synthesis

This chapter describes the following topics:

- [Synthesis Flow for the OpenSPARC T1 Processor](#)
- [Synthesis Output](#)

The scripts provided in the source code are for the Synopsys Design Compiler.

4.1 Synthesis Flow for the OpenSPARC T1 Processor

There are two types of synthesis scripts:

- One set to run the Synopsys Design Compiler (`rsyn` and `syn_command`)
- One set used as input for the Design Compiler

The main script used to run Synopsys Design Compiler is called `rsyn`. This is a PERL script that calls a second script, `syn_command`, once for each module you are synthesizing. The command-line options for the `rsyn` script are described in [CODE EXAMPLE 4-1](#).

CODE EXAMPLE 4-1 Command-Line Options for rsyn Script

```
rsyn : Run Synthesis for OpenSPARC T1

    -all
        to run synthesis for all blocks
    -h / -help
        to print help
    -syn_q_command='Your job Queue command'
        to specify Job queue command. e.g. specify submit command
        for LSF or GRID
    block_list :
        specify list of blocks to synthesize

Examples:

rsyn -all
rsyn fpu_add
```

Synthesis scripts for most of the modules are provided in the `$DV_ROOT/design` sub-directories. There are no synthesis scripts for the following types of modules:

- Megacell modules (SRAMS, TLB, TAG, Cache, etc.)
- Top-level hierarchical modules

Synopsys scripts, their locations, and their descriptions are listed in [TABLE 4-1](#).

TABLE 4-1 Synthesis Script Details

Script name	Location	Description
<code>run.scr</code>	<code>\$DV_ROOT/design/sys/synopsys/script</code>	Main synthesis script that calls <code>user_cfg.scr</code>
<code>project_sparc_cfg.scr</code>	<code>\$DV_ROOT/design/sys/synopsys/script</code>	SPARC module-specific synthesis script
<code>project_io_cfg.scr</code>	<code>\$DV_ROOT/design/sys/synopsys/script</code>	I/O module-specific synthesis script
<code>target_lib.scr</code>	<code>\$DV_ROOT/design/sys/synopsys/script</code>	Target library-specific script
<code>user_cfg.scr</code>	<i>Module directory</i> /synopsys/script	Module-specific synthesis script

The top-level Synopsys script, `run.scr`, calls the module-specific script named `user_cfg.scr`. The `user_cfg.scr` script calls the `project_sparc_cfg.scr` script or the `project_io_cfg.scr` script, depending on whether the module belongs to `sparc` or `io`.

The list of all modules with synthesis scripts is in the `$DV_ROOT/design/sys/synopsys/block.list` file. Each module has:

- A `synopsys` directory under the module directory
- A `script` directory under each `synopsys` directory
- The `user_cfg.scr` file under the `script` directory

For example, the `efc` module-specific synthesis script has the following directory path:

```
$DV_ROOT/design/sys/iop/efc/synopsys/script/user_cfg.scr
```

The target library is set to a generic library called `lsi_10k.db` in the `target_lib.scr` script. Modify this file to set your own target library and its required variables.

4.2 Synthesis Output

Running synthesis for a module creates files and directories under the *Module name/synopsys* directory, described in [TABLE 4-2](#).

TABLE 4-2 Synthesis Output

Name	Type	Description
<code>dc_shell.log</code>	File	Log file from running Design Compiler
<code>command.log</code>	File	Command log from running Design Compiler
<code>log</code>	Directory	Area report files from Design Compiler
<code>gate</code>	Directory	Gate netlist generated by Design Compiler
<code>.template</code>	Directory	Template directory used by Design Compiler

Design and Verification Manual Pages

This appendix provides the manual pages for commands used for OpenSPARC T1 design and verification.

A.1 `sims`

NAME

`sims` - Verilog rtl simulation environment and regression script

SYNOPSIS

`sims [args ...]`

NOTE: Use "=" instead of "space" to separate args and their options.

where args are:

SIMULATION ENV

`-sys=NAME`

`sys` is a pointer to a specific testbench configuration to be built and run. A config file is used to associate the `sys` with a set of default options to build the testbench and run diagnostics on it. The arguments in the config file are the same as the arguments passed on the command line.

`-group=NAME`

`group` name identifies a set of diags to run in a

regression. The presence of this argument indicates that this is a regression run. The group must be found in the diaglist. Multiple groups may be specified to be run within the same regression.

-group=NAME -alias=ALIAS

This combination of options gets the diag run time options from the diaglist based on the given group and alias. The group must be found in the diaglist. The alias is made up of diag_alias:name_tag. Only one group should be specified when using this command format.

VERILOG COMPILATION RELATED

-sim_type=vcs/ncv

Defines which simulator to use, vcs or ncverilog, Defaults to vcs.

-sim_q_command="command"

Defines which job queue manager command to use to launch jobs. Defaults to /bin/sh and runs simulation jobs on the local machine.

-ncv_build/-noncv_build

Builds a ncverilog model and the vera testbench. Defaults to off.

-ncv_build_args=OPTION

ncverilog compile options. Multiple options can be specified using multiple such arguments.

-ncv_use_vera/-noncv_use_vera

Compiles in the vera libraries. Defaults to off.

-vcs_build/-novcs_build

Builds a vcs model and the vera testbench. Defaults to off.

-vcs_build_args=OPTION

vcs compile options. Multiple options can be specified using multiple such arguments.

-vcs_clean/-novcs_clean

Wipes out the model directory and rebuilds it from scratch. Defaults to off.

-vcs_use_2state/-novcs_use_2state

Builds a two-state model instead of the default four-state model. This defaults to off.

-vcs_use_initreg/-novcs_use_initreg

Initializes all registers to a valid state (1/0). This feature works with -tg_seed to set the seed of the random

initialization. Defaults to off.

`-vcs_use_fsdb/-novcs_use_fsdb`
 Uses the debussy fsdb pli and include the dump calls in the testbench. this defaults to on.

`-vcs_use_vcsd/-novcs_use_vcsd`
 uses the vcs direct kernel interface to dump out debussy files. Defaults to on.

`-vcs_use_vera/-novcs_use_vera`
 Compiles in the vera libraries. If `-vcs_use_ntb` and `-vcs_use_vera` are used, `-vcs_use_ntb` wins. Defaults to off.

`-vcs_use_ntb/-novcs_use_ntb`
 Enables the use of NTB when building model (simv) and running simv. If `-vcs_use_ntb` and `-vcs_use_vera` are used, `-vcs_use_ntb` wins. Defaults to off.

`-vcs_use_rad/-novcs_use_rad`
 Uses the `+rad` option when building a vcs model (simv). Defaults to off.

`-vcs_use_sdf/-novcs_use_sdf`
 Builds vcs model (simv) with an sdf file. Defaults to off.

`-vcs_use_cli/-novcs_use_cli`
 Uses the `+cli` `-line` options when building a vcs model (simv). Defaults to off.

`-flist=FLIST`
 Full path to flist to be appended together to generate the final verilog flist. Multiple such arguments may be used and each flist will be concatenated into the final verilog flist used to build the model.

`-graft_flist=GRAFTFILE`
 GRAFTFILE is the full path to a file that lists each verilog file that will be grafted into the design. The full path to the verilog files must also be given in the GRAFTFILE.

`-vfile=FILE`
 Verilog file to be included into the flist

`-config_rtl=DEFINE`
 Places each such parameter as a 'define' in config.v to configure the model being built properly. This allows each testbench to select only the rtl code that it needs from the top-level rtl file.

-model=NAME

The name of a model to be built. The full path to a model is \$MODEL_DIR/\$model/\$vcs_rel_name.

-vcs_rel_name=NAME

Specifies the release of the model to be built. The full path to a model is \$MODEL_DIR/\$model/\$vcs_rel_name.

VERA COMPILATION RELATED

VERA and NTB share all of the vera options except a few. See NTB RELATED.

-vera_build/-novera_build

Builds the vera/ntb testbench. Defaults to on.

-vera_clean/-novera_clean

Performs a make clean on the vera/ntb testbench before building the model. Defaults to off.

-vera_build_args=OPTION

Vera testbench compile time options. Multiple options can be specified using multiple such commands. These are passed as arguments to the gmake call when building the vera testbench.

-vera_diag_args=OPTION

Vera/ntb diag compile-time options.
Multiple options can be specified using multiple such arguments.

-vera_dummy_diag=NAME

Provides a dummy vera diag name that will be overridden if a vera diag is specified, else used for vera diag compilation.

-vera_pal_diag_args=OPTION

Vera/ntb pal diag expansion options.
(i.e., "pal OPTIONS -o diag.vr diag.vrpal")
Multiple options can be specified using multiple such arguments.

-vera_proj_args=OPTION

Vera proj file-generation options. Multiple options can be specified using multiple such arguments.

-vera_vcon_file=ARG

Name of the vera vcon file that is used when running the simulation.

-vera_cov_obj=OBJ

This argument is passed to the vera Makefile as a OBJ=1 and to vera as -DOBJ to enable a given vera coverage object. Multiple

such arguments can be specified for multiple coverage objects.

NTB RELATED

NTB and VERA share all of the vera options except these:

-vcs_use_ntb/-novcs_use_ntb

Enables the use of NTB when building model (simv).

If -vcs_use_ntb and -vcs_use_vera are used, -vcs_use_ntb wins.

Defaults to off.

-ntb_lib/-nontb_lib

Enables the NTB two-part compile where the Vera/NTB files get compiled first into a libtb.so file which is dynamically loaded by vcs at runtime. The libtb.so file is built by the Vera Makefile, not sims. Use the Makefile to affect the build. If not using -ntb_lib, sims will build VCS and NTB together in one pass (use Makefile to affect that build as well). Defaults to off.

VERILOG RUNTIME RELATED

-vera_run/-novera_run

Runs the vcs simulation and loads in the vera proj file or the ntb libtb.so file. Defaults to on.

-vcd/-novcd

Signals the bench to dump in VCD format.

-vcdfile=filename

The name of the vcd dump file. If the file name starts with a "/", that is the file dumped to; otherwise, the actual file is created under \$tmp_dir/\$vcdfile and copied back to the current directory when the simulation ends. Use "-vcdfile='pwd'/filename" to force the file to be written in the current directory directly (not efficient since dumping is done over network instead of to a local disk).

-vcs_run/-novcs_run

Runs the vcs simulation (simv). Defaults to off.

-vcs_run_args=OPTION

vcs (simv) runtime options. Multiple options can be specified using multiple such arguments.

-vcs_finish=TIMESTAMP

Forces vcs to finish and exit at the specified timestamp.

`-fast_boot/-nofast_boot`
 Speeds up booting when using the ciop model. Passes the `+fast_boot` switch to the simv run and the `-sas_run_args=-DFAST_BOOT` and `-midas_args=-DFAST_BOOT` to sas and midas. Also sends `-DFAST_BOOT` to the diaglist and config file preprocessors.

`-debussy/-nodebussy`
 Enables debussy dump. This must be implemented in the testbench to work properly. Defaults to off.

`-start_dump=START`
 Starts dumping out a waveform after START number of units.

`-stop_dump=STOP`
 Stops dumping out a waveform after STOP number of units.

`-fsdb2vcd`
 Runs fsdb2vcd after the simulation has completed to generate a vcd file.

`-fsdbfile=filename`
 The name of the debussy dump file.
 If the file name starts with a `"/`, that is the file dumped to.
 Otherwise, the actual file is created under `$tmp_dir/$fsdbfile` and copied back to the current directory when the simulation ends.
 Use `"-fsdbfile='pwd'/filename"` to force the file to be written in the current directory directly (not efficient since dumping is done over network instead of to a local disk).

`-fsdbDumplimit=SIZE_IN_MB`
 Max size of Debussy dump file. Minimum value is 32MB.
 Latest values of signal values making up that size is saved.

`-fsdb_glitch`
 Turns on glitch and sequence dumping in fsdb file. This will collect glitches and sequence of events within time in the fsdb waveform. beware that this will cause the fsdb file size to grow significantly. This option effectively does this:
`setenv FSDB_ENV_DUMP_SEQ_NUM 1`
`setenv FSDB_ENV_MAX_GLITCH_NUM 0`
 Defaults to off.

`-rerun`
 Reruns the simulation from an existing regression run directory.

`-post_process_cmd=COMMAND`
 Post-processing command to be run after vcs (simv) run completes.

`-pre_process_cmd=COMMAND`

Pre-processing command to be run before vcs (simv) run starts.

`-use_denalirc=FILE`
Uses FILE as the .denalirc in the run area. Default copies
\$env_base/.denalirc

ZEROIN RELATED

`-zeroIn_checklist`
Runs 0in checklist

`-zeroIn_build`
Builds 0In pli for simulation into vcs model.

`-zeroInSearch_build`
Builds 0in search pli for simulation into vcs model.

`-zeroIn_build_args`
Additional arguments to be passed to the 0in command.

`-zeroIn_dbg_args`
Additional debug arguments to be passed to the 0in shell.

SAS RELATED

`-sas/-nosas`
Runs architecture-simulator. If vcs_run option is OFF,
simulation is sas-only. If vcs_run option is ON, sas
runs in lock-step with rtl. Defaults to off.

`-sas_run_args=DARGS`
Defines arguments for sas.

MIDAS RELATED

midas is the diag assembler.

`-midas_args=DARGS`
Arguments for midas. midas creates memory image and user-event
files from the assembly diag.

`-midas_only`
Compiles the diag using midas and exit without running it.

`-midas_use_tgseed`
Adds -DTG_SEED=tg_seed to midas command line. Use -tg_seed to
set the value passed to midas or use a random value from /dev/random.

SJM RELATED

sjm is the J-Bus bus functional model

-sjm_args

Arguments to be passed in to sjm_tstgen.pl for generation of an sjm random diagnostic.

-sjm/-nosjm

Generates a random sjm diagnostic using the -tg_seed if provided. Defaults to off.

-tg_seed

Random generator seed for sjm random test generators. Also the value passed to +initreg+ to randomly initialize registers when -vcs_use_initreg is used.

VCS COVERMETER RELATED

-vcs_use_cm/-novcs_use_cmd

Passes in the -cm switch to vcs at build time and simv at runtime. Defaults to off.

-vcs_cm_args=ARGS

Argument to be given to the -cm switch.

-vcs_cm_cond=ARGS

Argument to be given to the -cm_cond switch.

-vcs_cm_config=ARGS

Argument to be given to the -cm_hier switch.

-vcs_cm_fsmcfg=ARGS

Argument to be given to the -cm_fsmcfg switch. Specifies an FSM coverage configuration file.

-vcs_cm_name=ARGS

Argument to be given to the -cm_name switch. defaults to cm_data.

MISC

-nobuild

Master switch to disable all building options. There is no such thing as -build to enable all build options.

-copyall/-nocopyall

Copies back all files to launch directory after passing regression run. Normally, only failing runs cause a copy back of files. Defaults to off.

-copydump/-nocopydump
Copies back dump file to launch directory after passing regression run. Normally, only failing runs cause a copy back of non-log files. The file copied back is sim.fsdb, or sim.vcd if -fsdb2vcd option is set.
Default is off.

-tarcopy/-notarcopy
Copies back files using 'tar'. This only works in copyall or in the case the simulations 'fails' (per sims' determination).
Default is to use 'cp'.

-diag_pl_args=ARGS
If the assembly diag has a Perl portion at the end, it is put into diag.pl and is run as a Perl script.
This allows you to give arguments to that Perl script.
The arguments accumulate, if the option is used multiple times.

-pal_use_tgseed
Sends '-seed=<tg_seed_value>' to pal diags. Adds -pal_diag_args=-seed=tg_seed to midas command line, and -seed=tg_seed to pal options (vrpal diags). Use -tg_seed to set the value passed to midas or use a random value from /dev/random.

-parallel
When specifying multiple groups for regressions, this switch will submit each group to Job Q manager to be executed as a separate regression.
This has the effect of speeding up regression submissions.
NOTE: This switch must not be used with -injobq

-reg_count=COUNT
Runs the specified group multiple times in regression mode. This is useful when we want to run the same diag multiple times using a different random generator seed each time or some such.

-regress_id=ID
Specifies the name of the regression.

-report
Used to produce a report of a an old or running regression. With -group options, sims produces the report after the regression run. Report for the previous regression run can be produced using -regress_id=ID option along with this option.

-finish_mask=MASK
Masks for vcs simulation termination. Simulation terminates when it hits 'good_trap' or 'bad_trap'. For multithread

simulation, simulation terminates when any of the thread hits `bad_trap`, or all the threads specified by the `finish_mask` hits the `good_trap`.
example: `-finish_mask=0xe`
Simulation will be terminated by `good_trap`, if threads 1, 2 and 3 hit the `good_trap`.

-stub_mask=MASK

Mask for vcs simulation termination. Simulation ends when the stub driving the relevant bit in the mask is asserted. This is a hexadecimal value similar to `-finish_mask`.

-wait_cycle_to_kill=VAL

Passes a `+wait_cycle_to_kill` to the `simv` run. a testbench may chose to implement this plusarg to delay killing a simulation by a number of clock cycles to allow collection of some more data before exiting (e.g. waveform).

-rtl_timeout

Passes a `+TIMEOUT` to the `simv` run.
Sets the number of clock cycles after all threads have become inactive for the diag to exit with an error. If all threads hit good trap on their own the diag exits right away. If any of the threads is inactive without hitting good trap/bad trap the `rtl_timeout` will be reached and the diag fails. Defaults to 1000. This is only implemented in the `cmp` based testbenches.

-max_cycle

Passes a `+max_cycle` to the `simv` run.
Sets the maximum number of clock cycle that the diag will take to complete. Defaults to 30000. If `max_cycle` is hit the diag exits with a failure. Not all testbenches implement this feature.

-norun_diag_pl

Does not run `diag.pl` (if it exists) after `simv` (`vcs`) run.
Use this option if, for some reason, you want to run an existing assembly diag without the Perl part that is in the original diag.

-nosaslog

Turns off redirection of `sas` stdout to the `sas.log` file.
Use this option when doing interactive runs with `sas`.

-nosimslog

Turns off redirection of stdout and stderr to the `sims.log` file.
Use this option to get to the cli prompt when using `vcs` or to see a truncated `sim.log` file that exited with an error. This must be used if you want control-c to work while `vcs` is running.

`-nogzip`
Turns off compression of log files before they are copied over during regressions.

`-version`
Print version number.

`-help`
Prints this man page.

IT SYSTEM RELATED

`-use_iver=FILE`
Full path to iver file for frozen tools.

`-use_sims_iver`
For reruns of regression tests only, use `sims.iver` to choose TRE tool versions saved during original regression run.

`-dv_root=PATH`
Absolute path to design root directory. This overrides `$DV_ROOT`.

`-model_dir=PATH`
Absolute path to model root directory. This overrides `$MODEL_DIR`.

`-tmp_dir=PATH`
Path where temporary files such as debussy dumps will be created.

`-sims_config=FILE`
Full path to sims config file.

`-env_base=PATH`
Specifies the root directory for the bench environment. It is typically defined in the bench config file. It has no default.

`-config_cpp_args=OPTION`
Allows the user to provide CPP arguments (defines/undefines) that will be used when the testbench configuration file is processed through cpp. Multiple options are concatenated together.

`-result_dir=PATH`
Allows the regression run to be launched from a different directory than the one sims was launched from. Defaults to `$ENV{PWD}`.

`-diaglist=FILE`

Full path to diaglist file.

-diaglist_cpp_args=OPTION

Allows the user to provide CPP arguments (defines/undefines) that will be used when the diaglist file is processed through cpp. Multiple options are concatenated together.

-asm_diag_name=NAME

-tpt_diag_name=NAME

-tap_diag_name=NAME

-vera_diag_name=NAME

-vera_config_name=NAME

-efuse_image_name=NAME

-image_diag_name=NAME

-sjm_diag_name=NAME

-pci_diag_name=NAME

Name of the diagnostic to be run.

-asm_diag_root=PATH

-tpt_diag_root=PATH

-tap_diag_root=PATH

-vera_diag_root=PATH

-vera_config_root=PATH

-efuse_image_root=PATH

-image_diag_root=PATH

-sjm_diag_root=PATH

-pci_diag_root=PATH

Absolute path to diag root directory. sims will perform a find from here to find the specified type of diag. If more than one instance of the diag name is found under root, sims exits with an error. this option can be specified multiple times to allow multiple roots to be searched for the diag.

-asm_diag_path=PATH

-tpt_diag_path=PATH

-tap_diag_path=PATH

-vera_diag_path=PATH

-vera_config_path=PATH

-efuse_image_path=PATH

-image_diag_path=PATH

-sjm_diag_path=PATH

-pci_diag_path=PATH

Absolute path to diag directory. sims expects the specified diag to be in this directory. The last value of this option is the one used as the path.

ENV VARIABLES

sims sets or uses the following ENV variables that may be used with pre/post

processing scripts, and other internal tools:

TABLE A-1 Environment Variables

Environment Variable	Description
SIMS_LAUNCH_DIR	Path to launch directory where sims is running the job
ASM_DIAG_NAME	Contains the assembly diag name
VERA_LIBDIR	Dir where Vera files are compiled
DV_ROOT	Overwrite by -dv_root if specified
MODEL_DIR	Overwrite by -model_dir if specified
TRE_SEARCH	Based on -use_iver, -use_sims_iver
DENALI	User defined
VCS_HOME	User defined
VERA_HOME	User defined
NCV_HOME	User defined

PLUSARGS

+args are not implemented in sims. They are passed directly to simulator at compile time and at runtime.

DESCRIPTION

sims is the frontend for vcs to run single simulations and regressions.

How To Build Models

Build a vcs model using \$DV_ROOT as design root:

```
sims -sys=cmp -vcs_build
```

Build a ncverilog model using \$DV_ROOT as design root:

```
sims -sys=cmp -ncv_build
```

Build the vera testbench only using \$DV_ROOT as design root:

```
sims -sys=cmp -vera_build
```

Build a model from any design root:

```
sims -sys=cmp -vcs_build -dv_root=/home/regress/2002_06_03
```

Build a graft model from any design root:

```
sims -sys=cmp -vcs_build -dv_root=/model/2002_06_03 \  
-graft_flist=/regress/graftfile
```

Build a model and re-build the vera:

```
sims -sys=cmp -vcs_build -vera_clean
```

Build a model and turn off incremental compile:

```
sims -sys=cmp -vcs_build -vcs_clean
```

Build a model with a given name:

```
sims -sys=cmp -vcs_build -vcs_rel_name=mymodel
```

How to Run Models

Run a diag with default model:

```
sims -sys=cmp -vcs_run diag.s
```

Run a diag with a specified model:

```
sims -sys=cmp -vcs_rel_name=mymodel -vcs_run diag.s
```

Run a diag with debussy dump with default model:

```
sims -sys=cmp -debussy +dump=cmp_top:0 -vcs_run diag.s
```

=head2 Run regressions

Run a regression using \$DV_ROOT as design root:

```
sims -group=mini
```

Run a regression using \$DV_ROOT as design root and specify the diaglist:

```
sims -group=mini -diaglist=/home/user/my_dialist
```

Run a regression using any design root:

```
sims -group=mini -dv_root=/afara/design/regress/model/2002_06_03
```

A.2 midas

NAME

midas - assembles diags (Midas Is a Diag Assembler)

SYNOPSIS

midas [options] <diag_name>

DESCRIPTION

This program builds assembly diags. It is substantially more involved than simply assembling the diag because it also has to link the diag, program the MMU, and generate several output files.

The diag specified on the command line will be built. Pretty much everything else is configurable.

Options

The following are the options you need to get started:

-h Display man page.

-verbose [level] / -noverbose (abbreviated -v / -nov)
Sets verbosity level (default=2). -noverbose (or -nov) is a synonym for -verbose 0, which means to generate no output in the absence of errors. The highest level of verbosity currently defined is 3.

-version
Returns version information and exit.

-format
Displays help on the diag format and exit.

-config <file>
Uses this file as the config file instead of the one that is distributed with Midas.

-project <project>
Uses this project for project-specific configuration. Default is the environment variable \$PROJECT. Legal value is OpenSPARCT1.

Common Options

The following are the commonly used options:

- diag_root <path>
Uses the specified path as a base for finding standard include files. Default is \$DV_ROOT.
- build_dir <path>
Path (absolute or relative to where command is invoked) to directory where temporary files are generated and the build is done. Default is './build'.
- dest_dir <path>
Path (absolute or relative to where command is invoked) of where to store output files. Default is '.'.
- find_root <dir>
Interprets the diag on the command line as the name of a diag to search for. It does a breadth-first search under the specified directory. The default behavior is not to do any search, but to assume that the specified diag is a full or relative path to the file.
- find
This is a shortcut for "-find_root <diag_root>/verif/diag".
- mmu <mmu_type>
Generates programming for the specified MMU. Recognized options are "ultra2", "OpenSPARCT1".
- ttefmt <tte_format>
Specifies TTE format for those MMUs that require it. May be "sun4u" or "sun4v". Default is project specific: "sun4v" for OpenSPARC T1.
- tsbtagfmt <tsbtagfmt>
Specifies the format of the TSB tag. Legal values are 'tagaccess' and 'tagtarget'. Default is project-specific: 'tagaccess' OpenSPARC T1.
- force_build or -f
Builds the diag, even if it looks like it has the same input as before and the same args as before.
- copy_products / -nocopy_products
By default, the product files generated in the build directory are hard linked to the destination directory. The reason they are hard linked and not copied is for speed. If the hard link fails, it will fall back to a

copy in case the directories are on different physical disks. If `-copy_products` is given, however, it will always do a copy, not a hard link. Default is project specific: `-nocopy_products` for OpenSPARC T1.

`-E` Stops after the preprocessing stage.

`-cleanup / -nocleanup`
 If `-cleanup` is enabled, then after a successful build, the build directory is erased if and only if the build directory was created by this invocation of midas. Default is project specific: `-cleanup` for OpenSPARC T1.

`-force_cleanup / -noforce_cleanup`
 If `-cleanup` is enabled, but this invocation of midas did not create the build directory, `-force_cleanup` will remove the build directory anyway. Default is project specific: `-noforce_cleanup` for OpenSPARC T1.

`-D<symbol> or -D<symbol>=<value>`
 Adds a define to the preprocessing line. Option may be repeated.

`-stddef / -nostddef`
 Includes standard preprocessor definitions on command line. `-nostddef` disables these. Default is `-stddef`, but no standard symbols are currently defined.

`-I<dir>`
 Adds a directory to the include path used by cpp and m4. Path should be absolute or relative to the directory where midas was invoked. Option may be repeated.

`-stdinc / -nostdinc`
 With `-stdinc`, the standard include paths are used during preprocessing (both cpp and m4). `-nostdinc` disables these. The standard include directories are the directory where midas was invoked, the build directory and `<diag_root>/verif/diag/assembly/include` (keep in mind that `<diag_root>` defaults to `$DV_ROOT`). Default is `-stdinc`.

`-include_build / -noinclude_build`
 This option is only meaningful with `-nostdinc`. If standard includes are switched off, `-include_build` will add the build directory back to the include path. Default is `-noinclude_build`.

`-include_start / -noinclude_start`
 This option is only meaningful with `-nostdinc`. If

standard includes are switched off, `-include_start` will add the start directory (the directory where midas was invoked) back to the include path. Default is `-noinclude_start`.

`-L<dir>`

Adds a directory to the search path when looking for object files in a MIDAS_OBJ directive. Option may be repeated.

`-C<dir>`

Adds a directory to the search path when looking for C source files in a MIDAS_CC directive. Option may be repeated.

`-pal_diag_args <args>`

If the diag is run through pal, gives these arguments to the pal diag. Option may be repeated. Note that these arguments are given to the diag, not pal itself. For instance, "midas -pal_args -abc mydiag.pal -pal_diag_args def -pal_diag_args ghi" will run the pal command sline "pal -abc mydiag.pal def ghi".

`-build_threads <num_threads>`

When doing work that can be done in parallel (such as assembling a bunch of files), use `<num_threads>` to do it. Default is project specific: 3 for OpenSPARC T1.

`-print_errors / -noprint_errors`

If `-noprint_errors` is defined, then generation of error messages is turned off. When used with `-verbose 0`, midas is completely silent. This is probably only useful for the test harness (which is why the switch is there).

`-copy_products / -nocopy_products`

If this is set, then copies files from the build directory to the starting directory. With `-nocopy_products`, the files are hard linked instead. If it tries to create a hard link and fails, it will fall back to a copy. Default is `-nocopy_products`.

`-compress_image / -nocompress_image`

If `-compress_image` is enabled (as it is by default), then allows compressed mem.images to be generated. By default, all MMU-generated blocks are compressed when written to mem.image, meaning that instead of initializing unused sections to zero, they are simply uninitialized. The `-nocompress_image` is equivalent to explicitly putting a 'compressimage=0' in all

attr_text/attr_data blocks.

-env_zero / -noenv_zero

When compressing blocks, if -env_zero is enabled the blocks will contain '// zero_image' directives to the environment. These directives are supported only by OpenSPARC T1, and they are used to backdoor initialize large tracts of memory to zero. If -noenv_zero is used, then compression will simply leave the data uninitialized.

-default_radix <decimal|hex>

Radix to assume for all parameters that do not explicitly start with '0x'. Default is 'decimal'.

-gen_all_tsbs / -nogen_all_tsbs

If -gen_all_tsbs is given, then all TSBs that are defined are written to the memory image. If -nogen_all_tsbs, then generate only the TSBs that are used. Default is project specific: -nogen_all_tsbs for OpenSPARC T1.

-allow_tsb_conflicts / -noallow_tsb_conflicts

If -allow_tsb_conflicts is enabled, then it is legal to have multiple virtual addresses map to the same entry in a TSB. A linked list will be created to hold all entries. With -noallow_tsb_conflicts (which is the default for N1), collisions in the TSB can only happen with the same VA but different contexts. Default is project specific.

-allow_empty_sections / -noallow_empty_sections

If TEXT_VA is specified, then at least one attr_text block for the section has to be specified, and the same is true for DATA_VA and attr_data blocks. If -allow_empty_sections is specified, then midas will allow you to specify a TEXT_VA(DATA_VA) for the section, even if the section has no attr_text(attr_data) blocks. Of course, any text(data) in such a section will be ignored. Default is project specific: -noallow_empty_sections for OpenSPARC T1.

-allow_duplicate_tags / -noallow_duplicate_tags

When adding to a TSB link list, it is an error to add the same tag twice. -allow_duplicate_tags suspends the error check. Default is project specific: -noallow_duplicate_tags for OpenSPARC T1.

-allow_illegal_page_sizes / -noallow_illegal_page_sizes

If -allow_illegal_page_sizes, then tte_size attributes

are not checked for valid values, though they are still checked against the width of the field. For instance, in the OpenSPARC T1 MMU, there are 3 page bits, so values can be specified 0-7. However, the only legal values for OpenSPARC T1 are 0, 1, 3, and 5, and unless `-allow_illegal_page_sizes` is in effect, setting page bits of 2, 4, 6, or 7 will cause an error. The default is project specific: `-noallow_illegal_page_sizes` for OpenSPARC T1.

`-allow_misaligned_tsb_base / -noallow_misaligned_tsb_base`
If `-allow_misaligned_tsb_base` is set, then a TSB base address need not be aligned with the TSB size.
If an unaligned address is specified as the base and `-allow_misaligned_tsb_base` is specified, then midas will forcibly align the address. Default should be `-noallow_misaligned_tsb_base` for all projects.

`-errcode <error_code>`
Prints a one-line description for the midas error code, then exits with status 0.

Configuring Commands

midas runs several commands in the course of its operation. Several of these can be configured. The configurable commands are: `pal`, `cpp`, `m4`, `gcc`, `as`, and `ld`. Each configurable command has 3 associated options:

`-std_<command>_args / -nostd_<command>_args`
When `-std_<command>_args` is enabled, the standard set of arguments for `<command>` is used. Default is `-std_<command>_args`

`-<command>_args <args>`
Adds `<args>` to the argument list for the specified `<command>`.

`-<command>_cmd <custom_command>`
Uses `<custom_command>` to run the specified `<command>` instead of the standard version.

Example

For instance, to add `-foo` to the link line, use `my_cpp` to preprocess, and not use any standard assembler options, use:

```
midas -ld_args -foo -cpp_cmd my_cpp -nostd_as_args mydiag.s
```

Configuring Filenames

There are several generated files, and they all have default names. You can configure the names of many of the files with the following option:

`-file <tag>=<name>`
Causes midas to name the file whose tag is <tag> to be named <name> instead of the default. <name> is treated as the name of a file in the build directory.

Valid tags for the `-file` option are:

`src` Local version of the original source code for the diag.
Default is `diag.src`.

`s` Assembly portion of diag before any preprocessing.
Default is `diag.s`.

`pl` Perl portion of the diag. Default is `diag.pl`.

`cpp` Output of the C preprocessor. Default is `diag.cpp`.

`m4` Output of the m4 preprocessor. Default is `diag.m4`.

`ldscr`
Linker script. Default is `diag.ls_scr`.

`exe` Linked executable. Default is `diag*.exe` where * is application name.

`image`
Verilog memory image. Default is `mem.image`.

`events`
Events file Default is `diag.ev`.

`syntab`
Symbol table. Default is `symbol.tbl`.

`goldfinger`
Specification to goldfinger on how to create memory image. Default is `diag.goldfinger`.

`directives`
File to contain midas directives after section splitting. Default is `diag.midas`.

`cmdfile`

File to stash the midas command-line. Default is .midas_args.

oldcmdfile

File to move old command-line options. Default is .midas_args.old.

oldm4

File to stash m4 output of previous run. Default is .midas.diag.m4.old.

Running Specific Phases

The build process is broken into phases: setup, preprocess, sectioning, assemble, link, postprocess, copydest, cleanup. The default behavior is to run all phases. You can, however, restrict operation to a selected set of phases.

-start_phase <phase_name>

Starts with the named phase and run all subsequent phases.

-phase <phase_name>

Runs the specified phase. If any -phase or -start_phase option exists, then by default all phases are off (except for the ones that -phase and -start_phase switch on). You can have multiple -phase options.

-E This option (mentioned above, which runs the preprocessor only) is just a shortcut for "-phase setup -phase preprocess").

Keep in mind that running selected phases is caveat emptor. There are cases where phases expect data or files from previous phases.

Errors

When midas is unable to run correctly it will exit with one of the following error codes.

M_NOERROR (#0): No error.
M_MISC (#1): Miscellaneous error
M_CODE (#2): Error in midas code.
M_DIR (#3): Directory error.
M_FILE (#4): File error.
M_CMDFAIL (#5): Command failed.
M_SECSYNTAX (#6): Error in section syntax.
M_ATTRSYNTAX (#7): Error in attr syntax.
M_MISSINGPARAM (#8): Missing parameter.
M_ILLEGALPARAM (#9): Illegal parameter.
M_OUTOFRANGE (#10): Out of range.

M_NOTNUM (#11): Not a number.
M_VACOLLIDE (#12): VA collision.
M_PACOLLIDE (#13): PA collision.
M_DIRECTIVESYNTAX (#14): Directive syntax error.
M_GENFAIL (#15): File generation failed.
M_ASMFAIL (#16): Assembler failed.
M_CCFAIL (#17): C compiler failed.
M_LINKFAIL (#18): Linker failed.
M_CPPFAIL (#19): CPP failed.
M_M4FAIL (#20): M4 preprocessor failed.
M_BADCONFIG (#21): Bad configuration.
M_EVENTERR (#22): Event parsing error.
M_ARGERR (#23): Argument error.
M_NOSEC (#24): Undefined section.
M_BADTSB (#25): Bad TSB.
M_BADALIGN (#26): Bad Alignment.
M_EMPTYSECTION (#27): Empty section.
M_TSBSYNTAX (#28): Error in tsb syntax.
M_APPSNTAX (#29): Error in app syntax.
M_MEMORY (#30): Memory error.
M_GOLDFINGERPARSE (#31): Goldfinger parse error.
M_GOLDFINGERARG (#32): Goldfinger arg error.
M_ELF (#33): ELF error.
M_BADLABEL (#34): Bad label.
M_GOLDFINGERMISC (#35): Uncategorized goldfinger error.
M_GOLDFINGERVERSION (#36): Bad version of goldfinger
M_DUPLICATETAG (#37): Duplicate tags in TSB
M_BLOCKSyntax (#38): Error defining goldfinger BLOCK

A.3 goldfinger

NAME

goldfinger - Midas' partner for building diags

SYNOPSIS

goldfinger [options]

DESCRIPTION

Goldfinger is midas' partner. Goldfinger is implemented in C and uses libelf for efficient analysis of ELF files. In the new regime, midas builds a linked executable and a command file (i.e., a .goldfinger file), which are then processed by goldfinger. The final output files are produced by goldfinger. It is the

intention that end users never invoke goldfinger directly, but only through midas. Nevertheless, users may find a case where they need to build a diag in a very non-standard way, and goldfinger provides a lower-level interface.

Goldfinger is typically used twice in a normal build process:

Section splitting

"goldfinger -splitsec <diag_file>" is used to split a diag into multiple assembly files, one per section. All embedded midas directives are written to a separate file.

Extracting from executable file

After the executable file is linked, midas needs to extract a memory image and a symbol table. The options "goldfinger -in <cmd_file> -genimage -gentsbs -gensymtab" will generate these files based on the directives in <cmd_file>.

Options

The options recognized by goldfinger are:

- h Show usage.
- version
Print version number and exit.
- v or -verbose
Make it more chatty.
- d or -debug
Make it very chatty.
- silent
Say nothing unless there's an error.
- n or -nooutput
Do not write any output files (for debugging only).
- noprint_errors
Don't print any error messages (usually used with -silent). You can still tell there was an error by the exit status.
- prefix <string>
Prepend <string> to each line of normal output.

`-destdir <dir>`

All created files go in this directory (or a relative path from it). The directory specified can be absolute or relative from where goldfinger is invoked.

`-srcdir <dir>`

If any of the command files specify filenames with relative paths, start searching from this directory. Note that the command files themselves are always specified absolutely or relative to where goldfinger is run.

Section splitting options

The following functions are meaningful when splitting sections.

`-splitsec <file>`

Splits the specified file into sections and writes an assembly file for each. Writes all midas directives into a file that must be specified by the `-midasfile` option.

`-midasfile <file>`

When doing section splitting, write all midas directives into this file.

Linked executable options

The following options are meaningful when analyzing linked executables.

`-in <command_file>`

Analyzes linked executables based in the directives in `<command_file>` (also referred to as a `.goldfinger` file).

`-genimage`

Generates a memory image based on the linked executable. Goes to stdout unless `-imagefile` is also specified.

`-imagefile <file>`

If `-genimage` is also specified, then redirects output here instead of stdout.

`-gensymtab`

Generates a symbol table from the linked executable. Goes to stdout unless `-symtabfile` is also specified.

`-symtabfile <file>`
 If `-gensymtab` is also specified, then writes the symbol table here instead of stdout.

`-gentsbs`
 Generates TSB programming based on the object files. It is in mem.image format. It will go to stdout unless `-imagefile` is also specified.

`-allow_tsb_conflicts`
 If `-tsbgen` is also provided, then doesn't cause a fatal error if there is a collision in the TSB. Adds to the TSB_LINK area instead.

`-allow_duplicate_tags`
 If `-allow_tsb_conflicts` is enabled, you are adding elements to a TSB_LINK area, and you try to add the same tag more than once, it is normally an error. This option disables the error check. This option is not recommended, since the duplicate tag defines a translation that can never be used.

`-nocompress`
 Does not do compression of mem.image sections for any sections, regardless of what is in the imagespec file.

`-noenvzero`
 Does not use the backdoor environment initialization to zero during image compression.

COMMAND FILE SYNTAX

In the command file (i.e., the .goldfinger file), all keywords can be either all uppercase or all lowercase, but not mixed. All numbers are 64-bit numbers. They can be written as decimal (first digit is 1-9), octal (first digit is 0), or hex (begins with 0x). A boolean option can be set to a nonzero number (true) or 0 (false). If a boolean option is named, but not assigned to (e.g, "COMPRESS;" instead of "COMPRESS = 1;"), then it is assigned to 1.

The attrs file is a list of four types of objects at the top level. They can appear on any order:

```
PA_SIZE = num;

APP <name>
  app_lines
END APP
```

```

TSB <name>
    tsb_lines
END TSB

TSB_LINK <name>
    tsb_link_lines
END TSB_LINK

```

The PA_SIZE field is the only top-level attribute. It defines the size of a physical address in bits. The default is 40.

All types of block contain two attributes:

```

SRC_FILE = "file";
    File name where this block is originally defined. Used
    for error and debugging output.

```

```

SRC_LINE = num;
    Line number in SRC_FILE where this block is originally
    defined. Used for error and debugging output.

```

APP

An APP object contains a few parameters and a list of block objects. An APP names one linked executable (see ELF_FILE) and a list of blocks that describe what to do with that file. The APP syntax is:

```

APP <appname>

    SRC_FILE = "source file";
    SRC_LINE = <num>;
    ELF_FILE = "executable file";

```

```

BLOCK <name>
    block_attrs
END BLOCK

```

```

BLOCK <another_name>
    block_attrs
    BLOCK_TSB <name>
        block_tsb_attrs
    END BLOCK_TSB
END BLOCK

```

...

END APP

ELF_FILE = "executable file";
Names the linked executable file (relative to
srcdir) that will be processed by this APP object.

BLOCK

A BLOCK defines a section of a linked executable that should be treated the same way. It can take the following parameters:

SECTION_NAME = "name";
Name of the section (e.g., ".MAIN") where this block is defined (used for debugging and error reporting). Used only for error reporting.

SEGMENT_NAME = "name";
Name of segment within the section (e.g., "text") for which this block is defined. Used only for error reporting.

LINK_SECTION = "name";
ELF section name where this block should look in the executable.

VA = <num>;
START_LABEL = "label";
Optionally specifies that the block should start at a particular address or label. You can specify one or the other, but not both. If neither is specified, then the starting VA for the elf section is used. The starting address, however it is specified, must be page aligned if it is to be added to a TSB.

END_VA = <num>;
END_LABEL = "label";
Optionally specifies that the block should end at a particular address or label. You can specify one or the other, but not both. If neither is specified, the the ending VA for the elf section is used.

COMPRESS = num;
Boolean. If set, then compresses the output of this block in the image. Compression means that if an entire line (i.e., aligned 32 bytes) is zero, the line is skipped. If -noenvzero is enabled, the 32 bytes are simply uninitialized. Otherwise, the backdoor '// zero_bytes'

syntax is used to initialize the memory in the environment. The backdoor syntax is specific to Niagara, so other projects should either adopt it or use -noenvzero.

IN_IMAGE = <boolean>;

If this is defined and num is zero, then doesn't write this block to the memory image. It is still included in the symbol table.

PA = <num>;

Physical address to write to the image file. Also used in symbol table.

RA = <num>;

Real address. Used to write into the TSB and for the symbol table. Written as 'X' in the symbol table if this is not specified.

RA_EQ_VA = <boolean>;

Boolean. If set, then sets RA to VA (perhaps after VA is computed from a label). It is illegal to set both RA and RA_EQ_VA.

PA_EQ_VA = <boolean>;

Boolean. If set, then sets PA to VA (perhaps after VA is computed from a label). It is illegal to set both RA and PA_EQ_VA.

NO_END_RANGE_CHECK = <boolean>;

If this is set to a nonzero value, then does not do an error check to make sure that end_va is not off the end of the segment.

BLOCK_TSB <name>

A BLOCK may contain one or more BLOCK_TSB blocks (delimited by "BLOCK_TSB <name>" and "END BLOCK_TSB". A BLOCK_TSB definition names a TSB (see TSB objects below) that the block should add itself to. It also defines parameters about how the block should add itself.

BLOCK_TSB

A BLOCK_TSB object defines how to add its containing block to a TSB. The name of the BLOCK_TSB object is the name of the TSB object (see below) that the block should add itself to.

TAG_BASE = num;

Number to use as the basis for TSB tags in this

attr block. The virtual address is OR'd into the proper bit range in this number (using TAG_ADDR_BITS) to form the TSB tags.

DATA_BASE = num;

Basis for the TSB data entries in this attr block. The real address is OR'd into the proper bit range in this number (using DATA_ADDR_BITS) to form the TSB data entries.

START_RA = num;

Starting real address for this attr block. Must be page aligned.

PAGE_SIZE = num;

Page size used for computing number of TSB entries and for alignment checks.

VA_INDEX_BITS = hi : lo;

Bits of the virtual address used to index a TSB. This is independant of the TSB size. If the TSBs being used have non-zero size_bits, they will add the size_bits to the 'hi' value specified.

TAG_ADDR_BITS = hi : lo;

Bits of the TSB tag that should contain a portion of the virtual address.

TTE_TAG_ADDR_BITS = hi : lo;

Bits in the TSB tag that contain the VA.

DATA_ADDR_BITS = hi : lo;

Bits of the TSB data that should contain a portion of the real address.

TSB

Defines a TSB. The ATTR blocks define which TSBs they want to write to use, and this holds the address translations for them.

START_ADDR = num;

Physical address of where this TSB should live in memory.

NUM_ENTRIES = num;

Number of entries in the TSB. This can be computed from SIZE_BITS for a particular MMU, but goldfinger doesn't want to get in the business of interpreting procoessor-specific bit fields.

SIZE_BITS = num;
 Sizes bits from the config register. It is used in the index calculation.

SPLIT = binary;
 If set and num is non-zero, then makes this a split TSB.

LINKAREA = name;
 If there is a collision at any entry in the TSB, it can create a linked list. This parameter names a TSB_LINK object (see below) that should contain the linked list. If a collision occurs when -allow_tsb_conflicts is not set, however, a collision is a fatal error.

TSB_LINK
 This defines a link area. If there is a collision in the TSB, a linked list can be used to track the multiple entries. This is an object for containing that linked list.

START_ADDR = num;
 Physical address where the linked list should live.

EXIT STATUS

The exit status will be 0 if the command succeeds. If it fails, it will exit with a positive exit status. The error codes are identical between goldfinger and midas. See "midas -h" for the most up to date description of the errors.

A.4 regreport

NAME

regreport - regression report generator

SYNOPSIS

regreport <options> [<directory> [<list>]]

DESCRIPTION

regreport examines all regression *.log files for diags under regression directory and prints report. It is called by sims for each diag. User typically calls regreport to generate summary of regression by typing following :

regreport <regression_direcotry>

<options>: [default]

-1 [<diag_dir>]:

Prints report for the specified or current-directory diag

-regress <output_file> <directory>:

In regression mode, regreport writes summary status for finished diags to a file until all diags are finished. NOTE: if some diag doesn't produce status, regreport,1.73 will wait forever.

-sas_only

Verilog simulator will not run, sas only.

-regenerate

Regenerates the status.log files in the diag directories.
Call it from the parent dir of all diag runs e.g. 2004_04_04/

<directory> [<list>]

Prints report for all diags under <directory>. <list> is 0 or more of simulation system (testbench) names, such as 'core1', 'chip8', etc. When nothing specified, all systems are included.

-simline

Typically only 1000 last lines of sim.log will be examined.

-simline=NNN can increase or decrease this number

-full

The whole file will be processed in regreport -1 mode

-[no]printpassed

Does not print passed diags in detailed summary

-[no]vlog

Disables vlog run on failing diags. Enabled by default
If a diag fails we run vlog on it. This is good for automation.

-debug

Runs with debug on.

-summary

Prints only summary

-emailaddr=<e-mail address>

Gives an email address where regression status will be sent.

A.5 vlog

NAME

vlog - post process verilog log file

SYNOPSIS

```
vlog [logfile|path_to_sim.log] [-debug -h -ccx -l2 -dram -cycles -[no]sort] [-perf]
```

DESCRIPTION

vlog is called by regreport and user does not need call it directly. Supported command options are:

- ccx
Prints ccx related messages
- l2
Prints l2 related messages
- dram
Prints dram related messages
- h
Prints out this screen
- debug
Script debug.
- cycles
Prints the cycles and not the time
- sort
Sorts sim.log according to time stamps first [default is on]
- perf
Prints all kinds of performance data - I, D miss e.t.c.

Examples:

```
vlog -ccx -l2 -dram >! vlog.log  
vlog <my_path>/sim.log >! vlog.log
```

